AD-A085 488 ROYAL AIRCRAFT ESTABLISHMENT FARNBOROUGH (ENGLAND) F/G 14/2
A TIME-SHARING COMPUTER PROGRAM TO DRIVE UP TO THREE INDEPENDEN--ETC(U)
MAY 79 W WATT
UNCLASSIFIED RAE-TR-79054 DRIC-BR-70107 NL

ROYAL AIRCRAFT ESTABLISHMENT

*

Technical Report 79054

May 1979

A TIME-SHARING COMPUTER PROGRAM TO DRIVE UP TO THREE INDEPENDENT FATIGUE TESTING MACHINES

by

W. Watt

*

Procurement Executive, Ministry of Defence
Farnborough, Hants

18 DRIC 19 BR-70107

UDC 620.178.3.05 : 519.688 : 681.3.025 : 621.9-529

11 May 79

ROYAL AIRCRAFT ESTABLISHMENT

9 Technical ~~Report~~ TR 79054

Received for printing 16 May 1979

6 A TIME-SHARING COMPUTER PROGRAM TO DRIVE UP TO THREE INDEPENDENT FATIGUE TESTING MACHINES

by

10 W. Watt

12 32

SUMMARY

WG is a PDP-8/E computer program which offers a choice from seven resident software sequence generators to each of three independent hardware waveform controllers which drive electrohydraulic fatigue testing machines. The program is controlled by teletype commands and paper tape input data. Under program control, each controller may be run at a specified fixed frequency or fixed loading rate. Various analysis counts are kept which may be typed on command. One generator takes its sequence off magnetic tape and is restricted to one controller at a time. Another version of WG may be produced, which will run five waveform controllers and two magnetic tape units.

14 RAE-TR-79054

Departmental Reference: Structures YSE/B/0626

310450 Jm

LIST OF CONTENTS

1 INTRODUCTION

For some years in the Fatigue Research Laboratory of Structures Department a 3-channel computer controlled system[1] has provided the main source of waveforms to drive electrohydraulic fatigue machines. This system, originally based on a MINIC 1 computer, has been progressively developed over the years. In its latest form a PDP-8/E computer supplies three hardware 'waveform controllers' with sequences of peak and trough values defining loads to be applied in separate fatigue tests. The controllers generate analogue waveforms by interpolating half-cosines between the specified points. The frequency of generation of each half cycle is controlled by the computer program WG, enabling either a specified fixed frequency or a specified 'fixed rate' (frequency inversely proportional to amplitude) waveform to be generated independently for each controller. WG does not monitor the actual loads and how they compare with the demand loads. However, the fatigue machines are equipped for Null Pacing[1] and when this is selected, demanded peak and trough values are held if necessary until the required load is reached. Alternatively, a separate method of continuous monitoring can be used which indicates when fixed rate demand loads are not being fully achieved.

WG is controlled by teletype (TTY) commands and runs three completely independent waveform controllers, each of which can be fed by any one of the resident software 'sequence generators' Random[1], Fixed[1], Block, TWIST[2], FALSTAFF[3], Magtape or Calibration loading. Program input data for the selected generator for a particular controller is read into core from the fast paper tape reader (PTR), by TTY commands, before the generator is started by another command. The Magtape sequence is held on the magnetic tape unit (MTA) and is read sequentially to one controller via double buffers in core while the generator is running. This generator is available to only one controller at a time. Sequence generators keep analysis counts of work done and these may be typed on demand. WG is a stand-alone program in assembly language (PAL) and the only other software requirement is the manufacturers' standard binary loader BIN.

2 SOFTWARE SEQUENCE GENERATORS

Generators start by clearing their analysis counts and checking that essential data has been loaded (section 5). The different generators are described below. GAC are air-ground-air cycles representing landing, taxying and take-off loads.

054

2.1 Random loading

This waveform is produced by a 'live' sequence generator from a small amount of program input data and the pseudo random number generator

$$R_{n+1} = R_n(2^{13} - 3) - 1 \text{ Modulo } 2^{31}$$

where the old value R_n is overwritten by the new value R_{n+1}. Every call to this algorithm produces a different random number in the range 0 to $2^{31} - 1$ and the sequence length before it repeats is 2^{31}. R may be given a start value in the input data. Fig 1 shows a sample waveform and the input data (section 5) to produce it.

The waveform is described as asymmetric half cycle restrained. It is a series of half cycles (or gusts) alternating either side of zero load (*ie* the externally applied mean load), any one of which can be replaced at random by a GAC which (here) is a single large negative gust. The probabilities of occurrence of the different gust magnitudes (amplitude spectrum shape) are data defined with separate definitions for positive and negative gusts. Each call to a waveform controller produces a half cosine between sequence points and extra zero loads are inserted automatically where a GAC is preceded or followed by a negative gust. The sign of the first gust and of every gust following a GAC is chosen by a pseudo 'head or tail' from the next value of R.

The magnitude of the next gust, or its substitution by a GAC is determined by the next value of R and a 'one-dimensional random walk' along the appropriate 'weighting list' to find the number of levels crossed. There are separate weighting lists defining probabilities of occurrence of positive and negative gusts, supplied as program input data. A weighting list is a series of positive integers (*ie* whole numbers) $P_0 \ldots P_n \ldots P_\ell$ in ascending order of magnitude. When $R < P_0$ a GAC is substituted and when $P_0 \leqslant R < P_1$ the 'lowest load' is chosen. Every level crossed thereafter means that the load is increased by one increment. GAC and lowest and incremental load magnitudes for positive and negative gusts are defined in the program amplitude parameters (PAPS) also supplied as program input data. Obviously P_ℓ must be one greater than the maximum possible value of R and is entered in the weighting lists as 2147483648 which is 2^{31}. For correct functioning of the generator the minimum weighting list is the two items P_0, $P_\ell (\ell = 1)$ and the maximum allowed is 21 items ($\ell = 20$). In the data, weighting lists are headed by the list length ($\ell + 1$). When $P_0 = 0$ there are no GAC and when $P_n = P_{n-1}$ there are no loads at the nth level. For a given weighting list the assumption is that in every 2^{31} gusts the probabilities of occurrence at each level would be $P_0/2^{31} \ldots (P_n - P_{n-1})/2^{31} \ldots (P_\ell - P_{\ell-1})/2^{31}$. On this basis the user chooses weighting lists and PAPS values for his required distribution. PAPS also contain 'truncation levels' for both positive and negative gusts and these correspond to the suffix n in the weighting lists. For each list, there is no truncation when its specified truncation level is greater than or equal to ℓ.

2.2 Fixed loading

This generator outputs a 'fixed list' cyclically. Each waveform controller can have a different fixed list read into core as program input data. Fixed lists consist of up to 3455 sequence points headed by the list length. The waveform can be irregular. Fig 2 shows a sample waveform and the program input data (section 5) from which it was produced.

2.3 Block loading

Fig 3 shows a sample of this waveform with the program input data (section 5) from which it was produced. The flight list is executed cyclically and consists of a list of up to 2890 flight type numbers. The flight list is headed by the list length. There can be up to 10 different flight types identified by the digits 0-9. Each flight type consists of a number of blocks as defined in the block list which describes flight types implicitly in ascending order starting with type 0. If less than 10 types are defined the missing types are automatically given zero block counts. Each type is defined by a block count followed by that number of block definitions. Each flight type can have any number of blocks (including zero) provided the overall limit of 400 blocks is not exceeded. The block list is headed by the number of flight types described in the list. Flight and block lists must be supplied as program input data. Any flight types named in the flight list which have zero block counts in the block list are ignored when the flight list is executed and do not appear in the analysis count of flights done. Conversely the block list may define more types than appear in the flight list.

Blocks are defined by the items M, A, F and BN where M and A can be signed. A block consists of BN half cycles of constant magnitude A, alternating at fixed frequency F on either side of mean load M. The direction of the first half cycle in the block is defined by the sign of A where negative means downwards. Transitions from one block mean to the next block mean are half cosines at minimum frequency (1 Hz).

054

2.4 TWIST loading

Fig 4 shows the start of this waveform and the program input data used. Its derivation and generation are described in Ref 2. Like Random loading, its sequence of 797330 points is generated 'live' from a relatively small amount of data held in core as fixed program constants. Although generated live the sequence is effectively 'fixed' since the generator always starts the sequence with the same values in the pseudo random number generators. The fixed sequence is repeated cyclically. It consists of 4000 flights selected pseudo randomly from 10 different flight types. Flights are of different lengths and in each the waveform is symmetric half cycle restrained with the same number of peaks as troughs at some or all of 10 possible load magnitudes. In WG these magnitudes are 14, 23, 33, 43, 53, 62, 72, 81, 94 and 100 in waveform controller units (section 7). They are chosen to utilize the full range of controller output and conform with the load spectrum envelope in the Report[2].

GAC may be inserted in front of every flight and these are specified as program input data and consist of a list of up to 50 signed loads preceded by the list length which may be zero if no GAC are required.

2.5 FALSTAFF loading

The derivation and generation of this sequence is described in Ref 3. Generation is pseudo random but like TWIST the sequence is effectively 'fixed'. Unlike Random and TWIST, the FALSTAFF generator requires large data tables and is unacceptably slow to be used 'live' in WG. The 35966 point sequence was generated off-line and a technique (Appendix A) devised to pack it into only 8K of core memory in the form of condensed lists. These lists are assembled as program constants in WG and they are available to all three waveform controllers independently. The sequence is repeated cyclically and consists of 200 different flights each starting and ending with a few GAC which form part of the sequence. Sequence points are integers in the range 1-32. For better definition, these values are multiplied by three before being output in waveform controller units (section 7). Fig 5 shows the first two flights under constant loading rate and the program input data required (section 5).

2.6 Magtape loading

This sequence generator is available to only one waveform controller at a time. It starts by checking that the MTA is free and switched ON LINE. If the magnetic tape is not initially positioned at BOT (beginning of tape) it is rewound automatically. The required sequence is normally generated off-line on a more powerful computer (probably using a high level language like FORTRAN) and written directly on a magnetic tape in fixed length data blocks (Appendix B). Alternatively the sequence may be produced off-line by hand or by a computer which has paper tape equipment but no MTA. WG has a special routine (Appendix B) to transfer this data to a magnetic tape. This routine can be used at any time when the PTR and MTA are otherwise free and when not more than two waveform controllers are active. This restriction is due to the organisation of the program (section 3). MTA input/output is done by 'direct access' to core and does not tie up the computer central processor unit.

The sequence is executed cyclically. Data blocks are read sequentially into two core buffers, one block per buffer. As one buffer is being emptied the other buffer is filled (at a much faster rate). As soon as the last block is read, *ie* while the last two blocks are being output the tape is rewound ready to repeat the sequence. Alternatively the magnetic tape might hold a number of repeat sequences sufficient to ensure that the test is completed before a rewind is necessary (*eg* a tape can hold the equivalent of 500 FALSTAFF sequences). This would help to avoid excessive tape wear and obviate the possibility of a time gap when a load might have to be held to allow completion of a very long rewind. Magnetic tapes can only stand a limited number of rewinds and short sequences up to 3455 set points (two blocks) should be run under fixed loading (section 2.2).

2.7 Calibration loading

This is used to check hardware response. It produces a constant amplitude waveform with peaks and troughs at 60 and 30 load units respectively or -60 and -30 if inverted (section 4). This generator does not keep analysis counts but is otherwise similar in operation to the other generators.

3 GENERAL DESCRIPTION

WG swaps computer time cyclically, in short variable length segments, between three independent programs numbered 1-3. Each program has its own waveform controller (section 7) and looks after its own PTR or MTA input and controller or TTY output. Also, each has its own data area for control variables, program input data and analysis counts making the three programs completely independent although running concurrently. The interrupt system is not used; sharable devices (PTR, MTA and TTY) are protected by software flags so that they are only available to one program at a time. Controllers are serviced the next time their programs are in control of the computer after they have asked for more data. They are designed to call for and to accept data for the next sequence point while outputting the current analogue half cycle thus giving a half cycle leeway and helping to make WG 'real-time' as well as 'time-sharing'.

To save programming time and effort these techniques were carried over from the MINIC program when they were thought simpler to implement than an interrupt system. The original MINIC had only a small amount of core and no interrupt system. The method is quite adequate for the program requirements and computer processor (CPU) utilization is such that each program appears to react instantly to commands and to run continuously.

WG swaps computer time between the three programs in a section of code called the control loop (Fig 6). Programs can be active or inactive. Inactive programs stay in the control loop while active programs leave it for the job they are engaged on. Computer time is never allowed to 'hang' and active programs return to the control loop every time they are held up waiting for a 'device ready' flag. Program code execution is fast compared with hardware devices (section 9). While swapping, WG interrogates the TTY keyboard and the command decoder (CD) assembles and processes legal commands to start jobs on inactive programs or stop jobs on active programs. The maximum speed of the TTY keyboard and printer is 10 characters per second. It is worth noting that in 1/10 second there are from about 350 to 3225 passes through the control loop depending on which jobs if any are running.

The above technique means that each program can only be active on one job at a time. Jobs fall into one of the following three categories:

(a) Reading program input data from the PTR.

(b) Running a sequence generator.

(c) Typing an analysis of work done by the last generator.

Normally, data input and analysis output jobs are allowed to run to completion after which the program becomes inactive and available for another job. Sequence generators, however, run continuously and commands are provided (section 4) to STOP, HOLD and CONTINUE these jobs. The STOP command terminates a sequence generator and brings the load to zero (effectively the mean load applied externally to the fatigue machine) before making the specified program inactive. The HOLD command brings the load to the holding level for the particular generator and saves 'continue paramaters' before making the program inactive. Zero is the holding level for Random, Fixed and TWIST while Block loading holds at the current block mean and the other generators hold the last output load. The CONTINUE command reactivates a held sequence generator from the point at which it was held. These two commands allow an operator to interrupt a sequence generator in order to request its latest analysis to be typed. Similarly at this point he may read in fresh program input data providing this will not jeopardise the subsequent operation of the software generator. One program input data item is the number HCL. This is decremented after every sequence point output and when it reaches zero an automatic HOLD is executed. As shown in Fig 6, the STOP and HOLD commands set the stop flag belonging to the specified program. All sequence generators check these flags before every half cycle (or full cycle in TWIST and Calibration loading) but they have no effect on other job categories. A KILL command is provided which will stop any job instantly, release any devices allocated to the program and make the program inactive. KILL may be used to curtail a data read or analysis output job but is provided mainly as a means of recovery after a device malfunction or operator error, *eg* when a paper tape reads right through the PTR due to insufficient data. KILL is also used to make a program inactive when its waveform controller stops requesting another sequence point due to malfunction, specimen failure or external stop button (section 7). Note that STOP and HOLD are ineffective here since both these commands involve output to the waveform controller and acceptance of the output before the program is made inactive.

The Magtape generator (section 2.6) books the MTA by setting a software flag. This flag is cleared when the generator is stopped by the STOP or KILL commands but it is not cleared by the HOLD command and the MTA remains allocated to this job (and program). If it is not convenient to continue this generator, to release the MTA, an analysis should be requested for the relevant program and this should be killed before completion. The KILL command releases all held devices but may only be issued to an active program.

4 TELETYPE COMMANDS AND MESSAGES

TTY commands are summarised in Table 1 and consist of two characters, the first of which is the program number 1-3 to which the command refers and the second is the character @ or A-V which specifies the command. Any erroneous character causes the command decoder CD to revert to readiness to receive a new two character command. Stop commands to inactive programs, start commands to active programs, data read commands made while the PTR is busy and illegal continue commands are ignored completely. In any case, command characters themselves are not echoed on the printer making this device independent from the keyboard. This is so that commands may be entered on the keyboard while an analysis is being typed on the printer. However, all legal commands @-U cause an immediate response on the printer in the form of a 'command echo' (described below). Normally command V , if legal, responds immediately with the latest analysis of the last generator started on the relevant program but will be held up until the printer is free if given while another program is already typing its analysis. As shown in Table 1, commands fall into fairly logical groups. Commands @-C involve stopping or continuing a program and are described in section 3. Commands D-J start the different sequence generators. Commands K-M read data used by all generators and N-U read data for particular generators (section 5). Command V is described above and in more detail in section 6.

Command messages take priority over analysis output and may interrupt them temporarily. For this reason, all command messages are enclosed in angle brackets < > and always end by moving the carriage to the start of a new line. All command messages are listed in Table 1. In order, they consist of command echoes, interactive messages, error messages and the <DONE n> message. Most messages end with the program number n to which they refer. Stop command echoes from @-B say what the command has done. Start command echoes from C-U say what the command means and are followed by an interactive message which allows the operator to verify or cancel the command. Responses to interactive messages are the only keyboard characters which are echoed on the printer. Commands C-T are verified by typing Y (for YES) in response to the message <SURE? Any other response aborts the command but N (for NO) is recommended to keep the TTY log tidy. Command U is verified by the message <PTR READY? as described in Appendix B.

The inactive message <INVERT LOADS? is typed by all sequence generators before they commence loading. If the response is N (or any character other than Y) loads will be tensile (normal) and as defined by the data. If the response is Y loads will be compressive (inverted) and each output load (section 7) will have opposite sign to normal*. The rest of the command messages are described in section 5 and/or Appendix B.

5 PROGRAM INPUT DATA

Data read by commands K-T is listed in Table 2 and data read by command U is described in section 2.6 and Appendix B. All data items consist of integer

---

* This applies only for fatigue machines which apply a tensile load when the demand signal is negative. Some machines use the opposite convention and for these 'inverted' would mean tensile.

numbers, some of which may be signed, which are read as lists of one or more items and automatically stored in the correct data area as 6, 12, 24 or 36-bit numbers depending on the program number and the expected data. Variable length lists (jobs 0 and Q-T) are headed by list lengths as indicated in Table 2.

When a data item is being read all leading non-digits except MINUS are ignored. ERASE is always ignored wherever it occurs. Numbers can be of any length compatible with the permitted range (Table 2) and storage size. Leading zeros are permitted except in the list read by job S which is read by a different routine which takes every digit as a list item and ignores everything else. Thus this list (other than the list length) does not need number terminators. All other data items must have a terminator which is any non-digit character including NULL (blank tape) but excluding ERASE. These flexible formats allow non-digit captions (excluding MINUS) between numbers in the data tape as well as any number of punctuation and tabulation characters (see Figs 1 to 5).

All data read jobs except M, N and U set 'data-in' flags once they have accepted their data. See section 8 for initial flag settings and in particular the flag for scale data (job L). The relevant sequence generators check these flags and exit with the message <NO DATA n> if any necessary flag is not set. Data read jobs start by clearing their flags in case the data is faulty. In each program's data area, one sector is shared by data read in by jobs Q, R and S for Fixed and Block loading and as buffers by Magtape jobs I and U, so all these jobs set or clear relevant flags accordingly.

Job K reads frequency data (section 7) which is mandatory for all sequence generators except Block loading. This generator clears the frequency data-in flag since it overwrites this data with its own different frequencies. The scale factor (section 7) read by job L is mandatory for all sequence generators. HCL (section 3) does not have a data-in flag since any value in the store is valid data which can be given a start value by job M. It is used by all generators (except Calibration loading, which therefore does not have an automatic HOLD). Some operators start a sequence with a small number in HCL so that hardware etc can be checked before the automatic HOLD when the waveform may be continued (with a new value in HCL or with HCL = 0) or started again as necessary. Starting with HCL = 0 effectively means there will never be an automatic HOLD since there would need to be 2^{36} half cycles before HCL again reached zero. Similarly, the start value of R (section 2.1) read by job N does not have a data-in flag. However, for identical Random sequences the start value of

R as well as the rest of the data should be the same. The rest of the program input data read by jobs O-T (like N) apply to particular generators and is described in section 2.

Table 2 shows the permitted range of each data item and most of these are checked by the data read jobs, usually after the list has been read. The standard input routine checks every number to ensure it does not overflow its allocated storage size. Job R checks the compatibility between items M and A in each block. Data read by jobs O and P are also interdependent and to ensure resulting loads are within range, job D (see section 2.1) checks their compatibility before starting its sequence generation. Successful read jobs terminate with the message <DONE n> . Failed data checks cause the relevant job to exit with the message <BAD DATA n> .

6 ANALYSIS OUTPUT

Each sequence generator (except Calibration loading) starts by identifying itself to the calling program's analysis flag. Thus the analysis command V knows which analysis output routine is to be run. If no generator has been run on the specified program since WG was first loaded the job exits with the message <NO DATA n> . Fig 7 shows examples of analyses for the six sequence generators (there is no analysis for Calibration loading). The first analysis line identifies the relevant program and generator and the last line is always END. Most analysis counts are kept in 36-bit stores and are output as 11-digit numbers.

Each analysis gives the number of half cycles done (HCD) and the number left to do (HCL) before automatic HOLD. HCD is sufficient for the data defined Fixed and Magtape sequences. The latter analysis includes the latest maximum number (0-6) of retries to read a block which is developing parity errors (Appendix B). HCD is also enough for the Block and FALSTAFF analyses, but these include the number of flights completed to make it easier to establish the point reached in the sequence. From the Block analysis the current flight and position in the current block can be deduced from the data defined flight and block lists. Similarly, FALSTAFF executes its 200 flights in order and the originating report[3] lists the exact sequence flight-by-flight. This analysis also includes the number of half cycles into the current flight.

The originating TWIST report[2] gives details of each of the ten flight types but only lists the order of occurrence of the first 50 flights from the total of 4000 flights in the sequence. For this reason the TWIST analysis gives separate

054

counts for each flight type completed and also includes the type letter of the current flight. These flight counts are listed in the type order E, D, C, A, B, F, G, H, I, J (as given in the caption) which is the actual order in which they face pseudo random selection by the 'draw without replacement' method. Although the number of loads at each level for each flight type is known, the actual levels of the half cycles in the uncompleted flight are not known precisely. This is because for each new application of a given flight type the order of occurrence of its loads changes. It was not considered worthwhile to keep separate load counts (similar to the flight counts) for the current flight although this could be done for both positive and negative loads. This would require more program core space and extra loading on the CPU (section 9). Note that the TWIST analysis does not include GAC. These can be calculated from the input data and the number of flights started. Note also that in TWIST, STOP and HOLD commands given during GAC output are delayed until the GAC are completed.

The program input data for Random loading defines this spectrum statistically but the exact sequence is not known and the random number may be given any start value. For this reason the Random analysis includes counts of half cycles at every possible load level (depending on the input data) including GAC substitutions, together with their magnitudes.

7 HARDWARE

This consists of a PDP-8/E computer with extended arithmetic element KE8-E, TTY, PTR, single transport MTA, 24K of 12-bit core and 12-channel buffered digital I/O unit DR8-EA. The latter unit interfaces three waveform controllers through a special multiplexer-cum-optical isolator. Although manufactured elsewhere, the MTA is compatible with the TM8-E system described in the handbook[4].

The waveform controllers and interface were designed and built in Structures Department and operate with the DR8-EA 'unlatched'. They flag their readiness to receive data about the next sequence point and this consists of the following three words:

Word 1 = load magnitude plus sign bit. Range ±100 units.
Word 2 = frequency F . Range 1-100 Hz.
Word 3 = scale factor. Range 1-100%.

These values are sent to the controller as 7-bit binary integers with an eighth bit reserved for the sign of the load (0 = positive). The corresponding output from the controller is an analogue voltage in the range ±10 V in the form of a half cosine wave from the old to the new sequence point and taking $1/2F$ seconds.

The voltage is attenuated by the scale factor. This voltage drives a fatigue testing machine. In the event of excess load or excess travel (specimen failure) these machines return a signal to the controller which inhibits any more data requests. In the software, to help guard against spurious data requests due to spikes in the power supplies the flag is checked again after a delay of about 30 μs to ensure it is still UP. The flag (unless spurious) remains up until the three words have been received. These words are sent (with built-in software delay loops) at about 36 μs intervals without further flag checking.

All three controllers are serviced by calls to the completely re-entrant subroutine OPTP with the next load as argument. For Word 1, OPTP converts the load to controller format with due regard to the calling program's 'Invert Loads' flag (section 4). Thus only OPTP knows about inverted loads and the rest of the software (*eg* analysis counts) works as though the loads were always 'normal'. Next, OPTP sets up Word 2 by checking the program's frequency data (normally input by command K but set automatically by Block loading). This data consists of the two items FF and FV . FF is a flag and FF = 0 means 'fixed frequency' when the frequency (Word 2) takes the value FV . FF ≠ 0 means 'fixed loading rate' and for every output the frequency is calculated as the integral part of FV/PP where PP is the magnitude of the load change involved. If this result falls outside the permitted range 1-100 the limiting value is used. For every sequence generator (so far) in WG , Word 3 is always the program's input scale factor (job L). When the three words are assembled OPTP checks the relevant controller's data request flag as described above. If the controller is not ready the program is swapped by a return to the control loop (section 3 and Fig 6). When the relevant program is again in control of the computer the control loop sends it back to the same point in OPTP when the flag is tested again. I/O to the controllers is via the DR8-EA in the form of 12-bit words. On output one of the most significant three bits is used to direct the rest of the word to the appropriate controller and on input the same three bits carry the data request flags. Bits 0-2 address controllers 1-3 respectively.

Waveform controllers have an intermediate register between the input register which is filled by the computer and the output register which supplies the half cosine wave. So long as the input register has been supplied with the three data words and the data request flag is thus cleared, when the half cosine has been completed the registers will be moved up one step, the next half cosine will start and the data request flag will be set. This allows the sequence generator 1/2F seconds in which to supply the next sequence point data. If it is late there will be a gap in the waveform which will hold the load at the end of the current half cosine while the last output from the computer is still in the intermediate register. Thus the controller works one sequence point behind the sequence generator and when the software brings a waveform to a controlled HOLD (or STOP), it outputs the holding load (or zero load) twice.

Waveform controllers have a number of switches, red and green indicator lights and a 7-digit decimal display of full cycles output, *ie* pairs of half cosines. These give the machine operator, who may be some distance away from the TTY, some control and monitoring capabilities at his working point. They are also used for waveform controller diagnostics. There are three STOP push buttons marked SINGULARITY POINT, IMMEDIATE and MD CYCLE, each with a red lamp, which stop the waveform at the end of the current half cosine, immediately or at the next mid-point of a half cosine respectively. There are corresponding RUN push buttons, with green lamps, which cancel the stop condition and continue the interrupted waveform. The RESET push button clears all registers including the LOAD CYCLE COUNT display and it zeros the output voltage. RESET can act as an immediate STOP and has a red lamp. When both RUN lamps are green the data request is enabled. Controllers also have a selector switch to divide demand frequency by 10. The LOAD CYCLE COUNT will often display a larger number than half the analysis HCD because as well as the extra stopping output, some sequence generators have extra outputs which are not recorded as HCD, *eg* TWIST GAC. In fact many of these extra half cosines form only quarter cycles in the waveform.

8 LOADING AND STARTING WG

Loading and starting procedures are described in the handbook[5] which shows how to key in the appropriate read-in mode loader RIM and how to use it to load the standard binary loader BIN from the PTR. BIN should be loaded into field 0 where space is reserved for it by WG. The binary program and FALSTAFF constants files WG.BN and FALS.BN are loaded separately, in any order, from the PTR using BIN. The loader automatically selects the correct data fields. WG must then be started at location 0200 in field 0.

When WG is first loaded it starts with all programs inactive, all devices free and all data-in flags cleared except for the scale data-in flags. WG is assembled with this flag set for all three programs and with the scale set to the value 100. This is the normal scale setting and only has to be read in (job L) if a different value is required for a sequence generator on a specified program.

054

If the computer is switched off (or suffers a power failure), WG may be re-started (at 0200 in field 0) when power is restored since the computer memory is non-volatile. When WG is re-started INIT (Fig 6) ensures that all programs are again inactive and all devices are free but the data-in flags (and data) will be the same as when the computer stopped. Normally BIN, WG and FALS need to be reloaded only after routine computer maintenance when diagnostic programs destroy the contents of memory.

9 PROGRAM ENHANCEMENTS

Program code occupies field 0 , fields 1-3 are reserved as data areas for programs 1-3 respectively and fields 4 and 5 hold the FALSTAFF constants which are sharable by all three programs. Apart from the page reserved for BIN there are 2½ free pages in field 0 which could be used to hold yet another sequence generator.

Timing exercises on the slowest sequence generator (TWIST) show that when all three programs are running this generator at maximum frequency (100 Hz) then CPU utilization is about 55% and this could rise momentarily to nearly 100% if all three started a sequence simultaneously or 94% if they started a flight simultaneously. Fatigue tests are rarely run at more than 30 Hz and one proposed modification to WG is to increase the number of programs (and waveform controllers) to five. This would involve only minimal changes to the program code and an extra 8K of core. Fields 4 and 5 would then be the data areas for programs 4 and 5 and the FALSTAFF constants would be moved to the extra fields 6 and 7. An extra item of hardware would be required, consisting of a BCD decoder on the output of the DR8-EA so that the three address bits (section 7) could address all five controllers. No such modification is required on the input side since all 12 bits are available as data request flags. This proposal includes the addition of a second MTA.

Another proposal is to have an extra program number 4 (or number 6 if added to the above modification) devoted solely to analysis output for the other programs thus obviating the present necessity to interrupt a sequence generator in order to type its analysis (section 3).

054

Appendix A

The original FALSTAFF computer program[3] was not intended to be used as a live generator but to produce the sequence for paper or magnetic tape driven fatigue testing machines. A more efficient live generator[6] has been produced to run a single channel at up to nearly 100 Hz, and which requires only 2K of 16-bit core memory. This program is written in assembly language for a PDP-11/10 computer which is a more powerful machine than the PDP-8/E. This program would require some development effort if required to run more than one channel asynchronously and the matrix storage space (at least) would have to be duplicated for each channel. The method used in WG and described below is very simple to implement, it is very fast in operation and the core storage requirement remains virtually the same however many channels are being run.

PACKED FALSTAFF SEQUENCE

This 35966 point sequence consists of integers in the range 1-32 and is listed in Ref 3. In WG, lists A-H are program constants used to pack the sequence into 8K memory locations (actually 8127 12-bit memory words). They are assembled into the binary file FALS.BN. Table A1 below shows the number of items in each list, their byte size (*ie* number of bits) and core allocation. Lists are stored in ascending order of core locations and where more than one item is packed into a word the bytes are packed from left to right. No list crosses a field boundary. In a field, location addresses go from 0000 to 7777 (octal).

Table A1

Storage of FALSTAFF lists

List name	No. of items (decimal)	Byte size	Core words (decimal)	Start addr (octal)	Data field
A	34566	1	2881	0000	5
B	20692	1	1725	0210	4
C	12471	2	2079	3505	4
D	4734	3	1184	5501	5
E	366	4	122	7544	4
F	200	6	100	0000	4
G	25	12	25	0157	4
H	11	12	11	0144	4

The sequence consists of 200 different flights and there are 11 different flight lengths. Each flight starts with GAC values 8, 5, 7, 5 or 8, 6, 8, 6 and ends with GAC values 6, 8, 6. H is a list of flight lengths less GAC, stored in ascending order of magnitude as negative numbers. F is a list of indices (1-11) to list H for each flight in ascending order. The most significant bit (MSB) of each item in F is used to flag the GAC set, *ie* when MSB = 0 the flight starts with 8, 5, 7, 5 and when MSB is 1 it starts with 8, 6, 8, 6. Thus the sequence is reduced to 34566 alternate peaks and troughs. The reduced sequence was analysed and the pertinent information is shown in Table A2 opposite.

Column 1 is a list, in ascending order of magnitude, of all the positive ranges (trough-to-peak) in the reduced sequence and column 2 lists the number of occurrences of each positive range. It is obvious that the numbers in column 2 get rapidly smaller with larger ranges. In PAL it is very easy to store the FALSTAFF sequence in half words (6-bits) but this would require nearly 18K of core. However, it is also quite convenient to deal with 1, 2, 3 or 4-bit stores and a method was devised, using lists with these byte sizes to store the reduced sequence but this took just over 8K of core. This method involved both positive and negative ranges where the number of bits required to store each range (in sequence) increased with larger ranges. The total storage was reduced to less than 8K by discarding the negative ranges and by arranging all the possible trough values in order of frequency of occurrence as shown in column 3. This is list G (Table A1). Column 4 lists the number of occurrences of the corresponding trough values in column 3 and has a similar distribution to column 2. The method of storing the reduced sequence is described below. Note that the complete sequence consists of alternating peaks and troughs.

The items in lists A-E are produced by a scan of the reduced sequence for positive ranges and trough values alternately. See Table A1 for the byte sizes in these lists. For peaks, if the positive range is 3 the next list A item is made 1 and the peak has been stored and the next sequence point is examined (as a trough). Otherwise the list A item is made 0 to signify that list B is involved in storing this peak as follows. If the positive range is 4 the next list B item is made 1 (and the peak has been stored) and if not it is 0 and list C is involved. Thus positive ranges of 5, 6 or 7 involve 0s in both lists A and B and values 1, 2 or 3 respectively in list C . Similarly positive ranges 8-14 involve 0s in lists A-C and values 1-7 in list D while the remaining positive ranges 15-26 involve 0s in lists A-D and values of 1-12 respectively in list E . Thus the number of bits required to store a peak vary from 1 to 11 with 41% needing

Table A2

Peak-trough analysis

Positive range value	Number of positive ranges	Trough value (list G)	No. of troughs	List name	Total bits used per sequence point	Table index TS
3	7163	10	6711	A	1	1
4	3834	11	4387	B	2	2
5	1622	9	1941			3
6	1127	12	1445	C	4	4
7	886	13	716			5
8	715	8	543			6
9	554	14	511			7
10	417	15	327			8
11	333	16	234	D	7	9
12	198	17	135			10
13	155	18	69			11
14	140	19	37			12
15	81	7	36			13
16	76	20	23			14
17	44	6	18			15
18	17	5	17			16
19	14	21	12			17
20	3	4	6			18
21	2	22	4	E	11	19
22	0	23	3			20
23	0	1	2			21
24	1	2	2			22
25	0	24	2			23
26	1	3	1			24
-	-	25	1			25

054

only 1 bit and less than 2% needing 11 bits. Similarly for troughs, if the trough value is 10 the next list A item is 1 and so on according to the trough values in column 3 (list G): *eg* a trough value of 25 needs 0s in lists A-D and the value 13 in list E . Column 5 shows the highest list and column 6 the number of storage bits involved for each occurrence in columns 2 and 4. Thus list A has 34566 1-bit bytes (2881 core words) which is the length of the reduced sequence.

The sequence generator starts every 200 flights by setting pointers to the start of lists A-F. These pointers are moved on one byte after every extraction from the corresponding list. As described above, for each flight the next item from list F indicates the appropriate initial GAC values and via list H the number of reduced sequence points in the flight. For these points, which are peaks and troughs alternately, the value TS (Column 7, Table A2) is accumulated by extracting the next values from lists A-E successively until a non-zero item is found. TS is this item value plus the maximum item size of each list passed in place of their zero markers, *eg* when the next items from lists A-D are all zeros and the list E item is 9 (say) then TS = 1 + 1 + 3 + 7 + 9 = 21 . There are no zeros in list E . More commonly, if the list A item is 1 then TS = 1 . If the next sequence point is a peak then its value is the last trough value plus 2 plus TS. If it is a trough then its value is obtained from list G indexed by TS . All flights end with the same GAC. This procedure is repeated for each of the 200 flights when the sequence is repeated by resetting the list pointers.

There might be further refinements to the above technique or more economical methods of storing the sequence, but unless they reduce the storage area to 4K or less (thus saving an entire memory field) they would be of no use to WG . Obviously any refinement that produced an inordinate increase in the program code size would be worse than useless, *eg* packing list G into half words.

Appendix B

MAGTAPE DATA

B.1 Magtape format

The Magtape sequence is recorded in NRZI, 9-channel normal mode at 800 BPI with ODD parity (see Ref 4). In this mode each byte comprises a parity bit and 8 data bits. Data consists of sequence points in the range ±100 and 'data markers' which are described below. For sequence points the least significant 7 data bits hold the magnitude and the most significant data bit holds the sign (0 = positive and 1 = negative). Magtape data is in fixed length 1728-byte blocks, *ie* half the available buffer area allowed in core for each program. All Magtape sequences start at the beginning of a tape (BOT) and the last block should be followed by the end-of-file mark EOF although it is not strictly necessary for the sequence generator in WG . If detected it causes the generator to stop with the error message <MAGTAPE ERROR NO 4140 > . EOF is a special 1-byte block containing the octal number 023. For I/O, one 8-bit magnetic tape byte corresponds to one 12-bit core word.

The generator uses a data marker consisting of a full 8-bit byte (377 octal or 255 decimal) for two purposes. Firstly, the last word in the last block must be a data marker so that the sequence generator can recognise the last block without having to read EOF. Secondly, since the sequence (plus 1 for the above marker) is not likely to be exactly divisible by 1728, to cater for the remainder, the data marker (in any position except the first or last word) is used to denote a partly filled block. Partly filled blocks must be filled out with zeros to make up the full 1728 words. The generator (section 2.6) uses a double buffer system, one buffer per block, so that while one buffer is being emptied (to the waveform controller) the other buffer is being filled (by direct access to core from the MTA). To avoid gaps in the waveform, partly filled blocks should not contain less than about 30 sequence points thus allowing time for the next block to be read. Furthermore, to allow maximum time for rewinding, the last two blocks should ideally be full, *ie* containing 1728 and 1727 sequence points respectively. The rewind command is issued as soon as the last block is read and while the penultimate block is still being sent to the controller.

B.2 Paper tape data for job U

The Magtape sequence is generated off-line, normally directly on magnetic tape or alternatively on paper tape. WG has a routine (job U) to read the paper tape sequence and write it on a magnetic tape. Job U may be run on any free program when the MTA is free and ON LINE. The paper tape data must be in standard 1728 word blocks containing sequence points and data markers as required and the format like any other input data (section 5). On paper tape, data markers are the decimal number 255. Data may be split into any number of paper tapes of manageable length each containing an exact number of blocks. One block requires about 60 feet of paper tape and only 2.68 inches of magnetic tape! Intermediate paper tapes (but not the last) are terminated by an extra data marker after the last block on the tape.

The read job U starts with the interactive message <PTR READY? allowing the operator time to mount the paper tape in the PTR before he types Y for yes. Alternatively he may type N to abort the job. The routine which reads single blocks into a core buffer starts by reading the first data item only. If this is a data marker it is recognised as the end of an intermediate tape and the interactive message is repeated for the next paper tape. This extra data marker is not written on the magnetic tape. If the first item is not a data marker the rest of the block is read into the buffer. Next the buffer is checked to ensure every item is a sequence point (range ±100) or a data marker (255). Any detected data error kills the job with the message <BAD DATA n> where n is the program number (section 4). If accepted, the block is altered to magnetic tape format and then written on the magnetic tape. Magnetic tape blocks are written sequentially. Next the block is read back from the MTA and compared with the buffer to ensure it has been written correctly. If this check fails the job is killed with the message <MAGTAPE ERROR NO 4002> , otherwise the block reading routine is called to read the next block from the PTR. The last block is recognised, since its last item is a data marker, and after it has been written the magnetic tape file is closed with EOF and the job terminates with the message <DONE n>. The <PTR READY? and <DONE n> messages indicate that no error has been detected so far.

To check that the paper tapes have been correctly formatted and read correctly by the PTR, each paper tape is check summed. Every number on the tape up to and including the extra data marker (if any) is added (as it is read) into a 12-bit store which is allowed to overflow, *ie* the check sum is kept modulo 4096. Because of the method used to store negative numbers the check sum values of these consist of the magnitudes of the numbers plus one for each negative sign. The off-line program which generates the paper tapes must perform the same check sum for each tape and add it as the last item on the tape. These check sums are compared before each tape is accepted. The check sums are not written on the magnetic tape.

If, as recommended (section 2.6), the magnetic tape sequence is made up of a number of repeat sequences there should be two versions of the last block of a single sequence and both should hold the last 1726 sequence points. For intermediate sequences the last two items in the last block should be 255 and 0 . On paper tape this would be followed by the extra data marker and tape check sum. For the last sequence these last two items in the last block should both be data markers. Note that both versions of the last tape would then have the same check sum. The single sequence could then be read in as often as required using the former version of the last tape (which could hold only one block) and finally with the latter version of the last tape.

B.3 Magnetic tape cleanliness

Magnetic tapes should be kept clean and occasionally run through a tape cleaning machine. Also, before a tape is mounted on the transport, the tape guides and reading head should be cleaned to remove all dust and tape particles. Dirt causes parity errors and these kill the job (I or U) with the message <MAGTAPE ERROR NO 4200> . The sequence generator (job I) allows up to six retries to read a block which exhibits parity errors before finally killing the job. In its analysis this job keeps a record of the highest number of retries (0–6) to read a single block. The generator starts with this count at 0 . Note this is not a running count of all parity failures. Periodic analyses (section 6) will show if a magnetic tape is deteriorating or getting dirty. Job U does not allow any retries when writing a tape.

Table 1

COMMANDS AND MESSAGES

CMD	Message	Remarks
n@	<KILL n>	KILL the job (D-V) on prog n
nA	<STOP n>	STOP the generator on prog n at zero load
nB	<SAVE HOLD n>	HOLD the generator on prog n and save 'continue' parameters. Also occurs automatically when HCL = 0
nC	<CONTINUE n>	CONTINUE 'held' generator on prog n
nD	<RANDOM LOAD n>	Start Random loading on prog n
nE	<FIXED LOAD n>	Start Fixed loading on prog n
nF	<BLOCK LOAD n>	Start Block loading on prog n
nG	<TWIST LOAD n>	Start Twist loading on prog n
nH	<FALSTAFF LOAD n>	Start Falstaff loading on prog n
nI	<MAGTAPE LOAD n>	Start Magtape loading on prog n
nJ	<CAL LOAD n>	Start Calibration loading on prog n
nK	<READ FREQ n>	Read frequency data for next generator (except Block) on prog n
nL	<READ SCALE n>	Read scale factor for next generator on prog n
nM	<READ NO TO DO n>	Read HCL start value for next generator (except Cal) on prog n
nN	<READ RAND n>	Read random number start value for job D on prog n
nO	<READ RAN LIST n>	Read weighting lists for job D on prog n
nP	<READ RAN PAPS n>	Read amplitude parameters for job D on prog n
nQ	<READ FIXED DATA n>	Read fixed sequence for job E on prog n
nR	<READ BLOCK DATA n>	Read block list for job F on prog n
nS	<READ FLIGHT NOS n>	Read flight list for job F on prog n
nT	<READ TWIST DATA n>	Read GAC list for job G on prog n
nU	<READ MTA DATA n>	Read from PTR to MTA using prog n (job I data)
nV		Type analysis for last generator run on prog n
	<SURE? α>	Verify command (C-T). α = Y for YES or α = N to abort job
	<PTR READY? α>	Verify command U as above. Repeats for next tape.
	<INVERT LOADS? α>	Query from jobs D-J. α = Y for YES or N for NO (normal)
	<BAD DATA n>	Job K-U or D on prog n killed by data checking routine
	<NO DATA n>	Job D-J or V on prog n killed - missing data
	<NO MAGTAPE n>	Job I or U on prog n killed - MTA not ON LINE or busy
	<MAGTAPE ERROR NO X>	Job I or U killed after MTA malfunction. X is 4-digit octal contents of MTA main status register
	<DONE n>	Successful end of data read job on prog n

n is the program number 1-3

α is the operator's response to an interactive message

Table 2

PROGRAM INPUT DATA

CMD	Item	Range	Comments
K*			Frequency data for all generators except Block loading:
	FF	0 or 1	Flag. 0 = Fixed frequency, 1 = Fixed loading rate
	FV	1 to 100 or	When FF = 0, FV = Fixed frequency
		1 to $(2^{24}-1)$	When FF = 1, Frequency = FV/PP for each transition where PP is the magnitude of the load change
L*			Scale factor required by all generators:
	S	1 to 100	S = 100 for ±10 V output from waveform controller
M*	HCL	0 to $(2^{36}-1)$	Start value of number of half cycles left before automatic hold. Used by all generators except Calibration loading
N*	R	0 to $(2^{31}-1)$	Start value of random number for Random loading
O			Weighting lists for Random loading:
	LP	2 to 21	Length of +ve list. LP = x + 1
	P_0	≥ 0	GAC level. Load = GAC (see PAPS)
	P_1	$> P_0$	Lowest level. Load = LPL
	P_2	$> P_1$	Second level. Load = LPL + LPI
	:		
	P_n	$> P_{n-1}$	nth level. Load = LPL + (n-1) LPI
	:		
	P_x	2^{31}	Highest level always 2147483648. Load = LPL + (x-1) LPI
	LN	2 to 21	Length of -ve list
	:		*Neg list similar to +ve list (could be identical)*
	N_y	2^{31}	Highest -ve level. Load = -(LNL + (y-1) LNI)
P			Program amplitude parameters (PAPS) for Random loading:
	GAC	-1 to -100	Air-ground-air load
	LPL	0 to 100	Magnitude of lowest +ve load
	LPI	0 to 100	Magnitude of +ve incremental load
	PT	1 to x	Truncation level for +ve loads
	LNL	0 to 100	Magnitude of lowest -ve load
	LNI	0 to 100	magnitude of -ve incremental load
	NT	1 to y	Truncation level for -ve loads (except GAC)
Q			Sequence for Fixed loading:
	L	1 to 3455	Length of list
	C_1 to C_L	-100 to 100	L signed sequence load points
R			Block list for Block loading:
	N_1	1 to 10	Number of flight cases. Each case defines a flight type in ascending order (0 to 9). One case follows:
	N_2	0 to 400	Number of blocks in this flight case (maximum total blocks = 400). N_2 blocks must follow.
			One block comprises:
	M	-100 to 100	Mean load for this block
	A	-100 to 100	Magnitude of alternating ½ cycles about M. Sign of A denotes direction of first ½ cycle (+ve = upwards)
	F	1 to 100	Fixed frequency for this block
	BN	1 to $(2^{24}-1)$	Number of ½ cycles in this block
S			Flight list for Block loading:
	N	1 to 2890	Number of flights in list
	T_1 to T_N	0 to 9	List of N flight type numbers
T*			GAC list for Twist loading:
	G	0 to 50	Number of load points in list
	K_1 to K_G	-100 to 100	List of G load points (usually negative)

* These data could be changed during a hold before a continue

REFERENCES

No.	Author	Title, etc
1	F. Sherratt P.R. Edwards	The use of small on-line computers for random loading fatigue testing and analysis. Journal of the Society of Environmental Engineers, December 1974
2	J.B. de Jonge D. Schütz H. Lowak J. Schijve	A standardized load sequence for flight simulation tests on transport aircraft wing structures. NLR TR 73029C, LBF-Bericht FB-106
3	H. Lowak M. Hück D. Schütz W. Schütz	Standardisiertes Einzelflugprogramm für Kampfflugzeuge FALSTAFF. LBF-Bericht No 3045 (1976), IABG-Bericht No TF 568 (1976)
4	-	PDP-8/E Small Computer Handbook PDP-8 Handbook Series Digital Equipment Corporation
5	-	Introduction to Programming PDP-8 Handbook Series Digital Equipment Corporation
6	J.H. Argyris W. Aicher H.J. Ertelt	Analyse und Synthese von Betriebsbelastungen. ISD-Bericht No 193 (1976)

```
RAND DATA
0,67 CONSTANT FREQUENCY
50   SCALE
200  NO TO DO
2    START RAN NO
WEIGHTING LISTS
5 POS LIST LENGTH
0107374182 GAC = V%
1395864371 LOWEST = LX%
1932735283 NEXT = XXV%
2104533975 NEXT = VIII%
2147483648 LAST = II%
4 NEG LIST
0107374182  V%
1503238554  LXV%
1932735283  XX%
2147483648  X%
PAPS
-50 GAC LEVEL
 10 LOW POS LEVEL
 10 POS STEP
  5 NO TRUNCATION
 12 LOW NEG LEVEL
 12 NEG STEP
  4 NO TRUNCATION
```

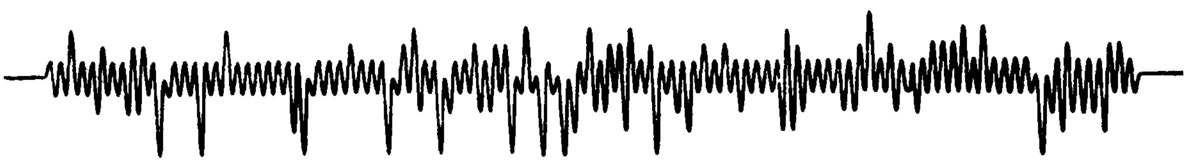


Fig 1   Random loading

```
FIXED DATA
0,40 CONSTANT FREQ
50 SCALE
98 NO TO DO
14 LIST LENGTH
-30,-10,-40,0,-30,-10,-30
30,10,40,0,30,10,30
```

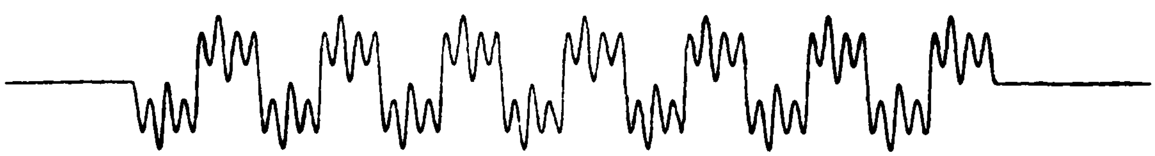


Fig 2   Fixed loading

```
BLOCK DATA
100 SCALE
140 NO TO DO
2 FLIGHT LIST LENGTH
1,2 FLIGHT LIST
3 CASES IN BLOCK LIST
0 NUL FLIGHT 0
2 BLOCKS IN FLIGHT I
10 20 20 4
10 -10 30 12
3 BLOCKS IN FLIGHT II
10 5 100 50
15 10 50 24
20 5 100 50
NUL FLIGHTS III TO IX
```

Fig 3 Block loading

```
TWIST DATA
0,25 CONSTANT FREQ
100 SCALE
300 NO TO DO
3 GAC LIST
-65,-55,-65
```

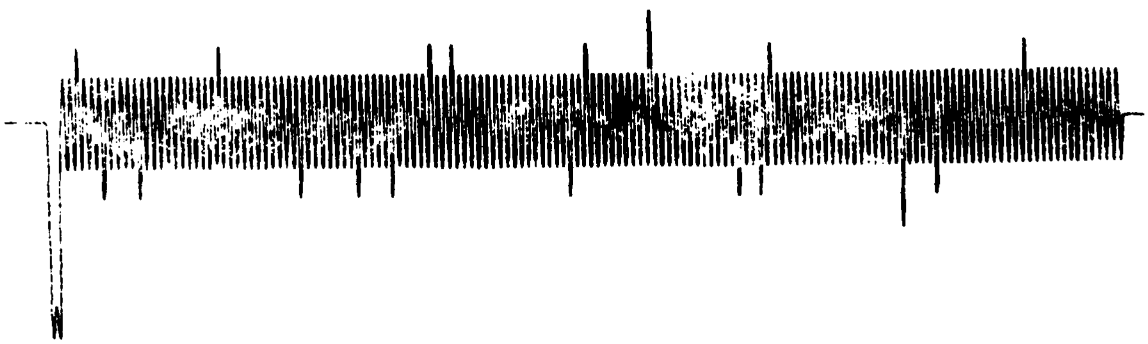


Fig 4 Twist loading

```
FALSTAFF DATA
1,400 CONSTANT LOADING RATE
100 SCALE
506 NO TO DO
```

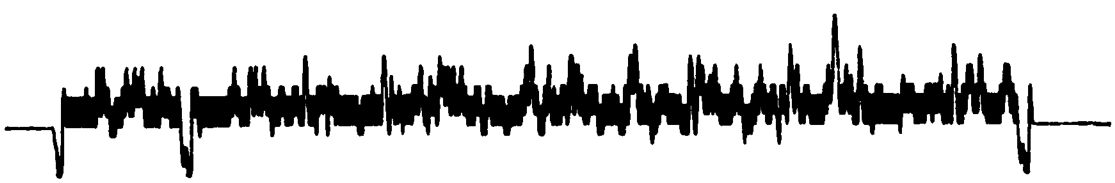
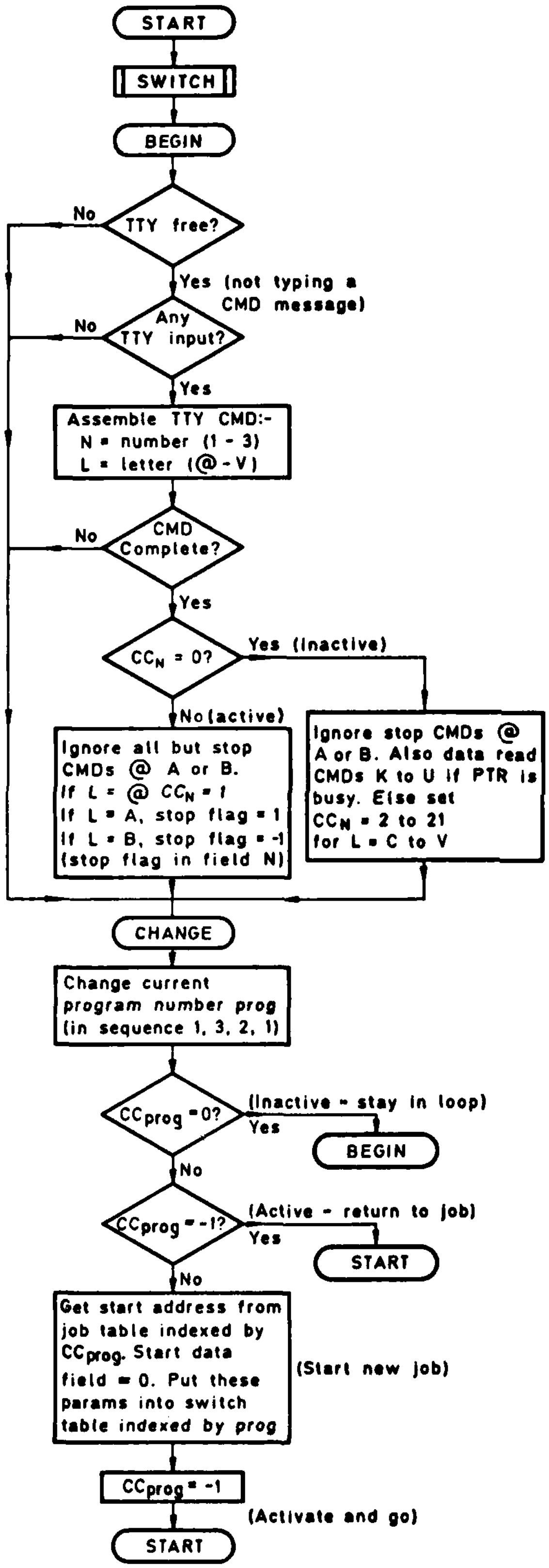
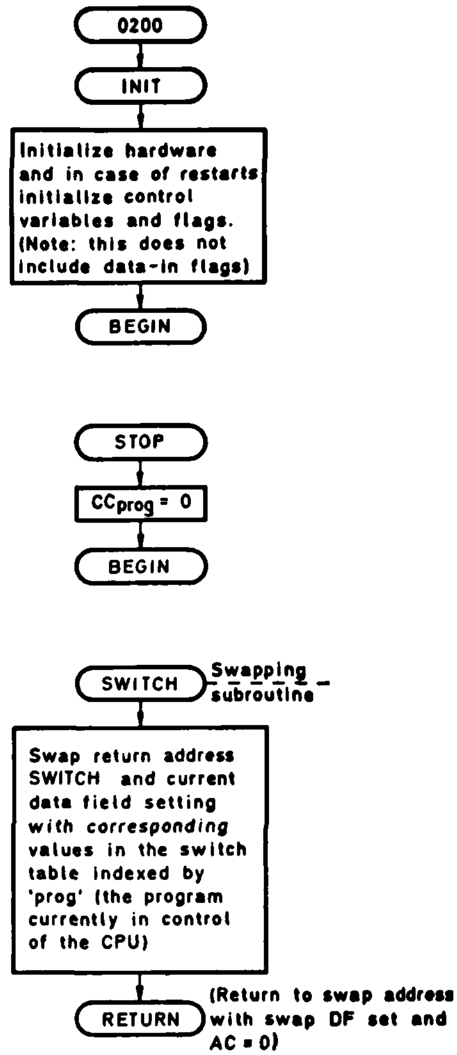


Fig 5 Falstaff loading





A program's first call to SWITCH (at START) takes WG to the start of a job where it will eventually meet another call to SWITCH (usually when baulked by I/O) which returns WG to BEGIN — and so on for each 'active' program.

When a sequence generator "holds" the *continue* address and data field are saved. The "continue" job (command C) places the saved parameters in the switch table and returns to the interrupted generator by a jump to START.

TR 79054

Fig 6 Control loop

Fig 7

```
CHANNEL 1 RANDOM ANALYSIS

1/2 CYCLES DONE    00000010000
1/2 CYCLES LEFT    00000000000
LAST RANDOM NO     00358299812
G.A. CYCLES (+VE) 00000000234
G.A. CYCLES (-VE) 00000000215
G.A. LEVEL -060

+VE LEVEL   1/2 CYCLES
   015    00000003249
   025    00000000967
   035    00000000218
   045    00000000331

-VE LEVEL   1/2 CYCLES
   012    00000003065
   020    00000000939
   028    00000000344
   036    00000000224
   044    00000000214
END

CHANNEL 2 FIXED ANALYSIS

1/2 CYCLES DONE    00000009301
1/2 CYCLES LEFT    00000990699
END

CHANNEL 3 BLOCK ANALYSIS

1/2 CYCLES DONE    00000000100
1/2 CYCLES LEFT    00000000000
FLIGHTS COMPLETED 00000000001
END

CHANNEL 1 TWIST ANALYSIS

1/2 CYCLES DONE    00000100000
1/2 CYCLES LEFT    00000000000
FLIGHTS COMPLETED EDCABFGHIJ
   00000000002
   00000000002
   00000000000
   00000000000
   00000000000
   00000000006
   00000000018
   00000000067
   00000000152
   00000000253
CURRENT FLIGHT C
END

CHANNEL 2 FALSTAFF ANALYSIS

1/2 CYCLES DONE    00000035970
1/2 CYCLES LEFT    68719440766
FLIGHTS COMPLETED 00000000200
1/2 CYCLES INTO FLIGHT     004
END

CHANNEL 3 MAGTAPE ANALYSIS

1/2 CYCLES DONE    00000001000
1/2 CYCLES LEFT    00000000000
MAX PARITY RETRIES 0
END
```

Fig 7 Sample analyses

REPORT DOCUMENTATION PAGE

Overall security classification of this page

UNLIMITED

As far as possible this page should contain only unclassified information. If it is necessary to enter classified information, the box above must be marked to indicate the classification, e.g. Restricted, Confidential or Secret.

1. DRIC Reference (to be added by DRIC)	2. Originator's Reference RAE TR 79054	3. Agency Reference N/A	4. Report Security Classification/Marking UNLIMITED
5. DRIC Code for Originator 7673000W	6. Originator (Corporate Author) Name and Location Royal Aircraft Establishment, Farnborough, Hants, UK		
5a. Sponsoring Agency's Code N/A	6a. Sponsoring Agency (Contract Authority) Name and Location N/A		

7. Title A time-sharing computer program to drive up to three independent fatigue testing machines

7a. (For Translations) Title in Foreign Language

7b. (For Conference Papers) Title, Place and Date of Conference

8. Author 1. Surname, Initials	9a. Author 2	9b. Authors 3, 4	10. Date	Pages	Refs.
Watt, W.			May 1979	30	6

11. Contract Number	12. Period	13. Project	14. Other Reference Nos.
N/A	N/A		

15. Distribution statement

(a) Controlled by –

(b) Special limitations (if any) –

16. Descriptors (Keywords) (Descriptors marked * are selected from TEST)

Fatigue. Computer. Program.

17. Abstract

WG is a PDP-8/E computer program which offers a choice from seven resident software sequence generators to each of three independent hardware waveform controllers which drive electrohydraulic fatigue testing machines. The program is controlled by teletype commands and paper tape input data. Under program control, each controller may be run at a specified fixed frequency or fixed loading rate. Various analysis counts are kept which may be typed on command. One generator takes its sequence off magnetic tape and is restricted to one controller at a time. Another version of WG may be produced, which will run five waveform controllers and two magnetic tape units.

RAE Form A143